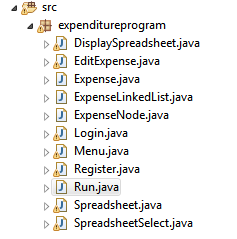
Appendix A: Source Code

List of classes :



**Run.java**

**package** expenditureprogram;

**import** javax.swing.SwingUtilities;

**public** **class** Run {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

SwingUtilities.*invokeLater*(**new** Runnable() {

**public** **void** run() {

**new** Login();

}

});

}

}

Login.java

package expenditureprogram;

import java.awt.Font;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.IOException;

import java.io.RandomAccessFile;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JOptionPane;

import javax.swing.JPanel;

import javax.swing.JPasswordField;

import javax.swing.JTextField;

public class Login extends JFrame {

// declaration of instance fields

private JLabel userLabel;

private JLabel passwordLabel;

private JLabel title;

private static JTextField userText;

private JPasswordField passwordText;

private JButton loginButton;

private JButton registerButton;

private static JPanel loginPanel;

// constructor

public Login() {

// sets the title, size, and close operation of the window

super("Expenditures Program");

setSize(500, 500);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

// initializing the panel and adding it to the window

loginPanel = new JPanel();

loginPanel.setLayout(null);

add(loginPanel);

// Initializing buttons, text fields, and labels

title = new JLabel("Login");

title.setBounds(200, 10, 80, 25);

title.setFont(new Font("Verdana", 1, 20));

userLabel = new JLabel("Username");

userLabel.setBounds(125, 80, 80, 25);

userText = new JTextField(20);

userText.setBounds(200, 80, 165, 25);

passwordLabel = new JLabel("Password");

passwordLabel.setBounds(125, 160, 80, 25);

// changes the password into dots, hides the password information

passwordText = new JPasswordField();

passwordText.setBounds(200, 160, 165, 25);

registerButton = new JButton("Register");

registerButton.setBounds(270, 240, 100, 25);

registerButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

// if the user is not registered, then the user is redirected to the register section

if (!userRegistered()) {

userText.setText("");

passwordText.setText("");

loginPanel.hide();

Register newUser = new Register();

// adds the panel onto the JFrame

add(newUser);

// otherwise, the user already has an account and can proceed to login

} else {

JOptionPane.showMessageDialog(null, "You already registered! \nPlease login.", "User already registered",

JOptionPane.NO\_OPTION);

}

}

});

loginButton = new JButton("Login");

loginButton.setBounds(125, 240, 80, 25);

loginButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

// if the username and password match, the login screen will be disposed and the user enters the menu

if (checkCredentials()) {

loginPanel.hide();

new Menu();

setVisible(false);

// otherwise, the user entered invalid login credentials

} else {

JOptionPane.showMessageDialog(null, "Invalid login credentials! \nPlease try again.", "Login Error",

JOptionPane.ERROR\_MESSAGE);

}

}

});

// adds all text fields, labels, and buttons to the panel

loginPanel.add(title);

loginPanel.add(userLabel);

loginPanel.add(userText);

loginPanel.add(passwordLabel);

loginPanel.add(passwordText);

loginPanel.add(registerButton);

loginPanel.add(loginButton);

// sets the frame to be visible

setVisible(true);

}

/\*\*

\* Method that determines whether the user has registered or not

\* @return true if the text file exists, false if the text file does not exist

\*/

public static boolean userRegistered() {

try {

final RandomAccessFile users = new RandomAccessFile("users.txt", "rw");

// if the text file does not exist, then the user has not registered and return false, otherwise return true

if (users.length() == 0L) {

return false;

} else {

return true;

}

} catch (IOException e) {

e.getMessage();

}

return false;

}

/\*\*

\* Method that determines whether the user's username and password match

\* @return true if the username and password match the credentials saved in the text file, otherwise return false

\*/

public boolean checkCredentials() {

boolean loggedIn = false;

try {

RandomAccessFile credentials = new RandomAccessFile("users.txt", "r");

String readFile = credentials.readLine();

String name = readFile.substring(0, readFile.indexOf("\\"));

String savedPassword = readFile.substring(readFile.indexOf("\\") + 1);

String password = "";

//

for (int i = 0; i < savedPassword.length(); ++i) {

int j = savedPassword.charAt(i) + '\u0003';

password += (char)j;

}

if (name.equals(userText.getText()) && password.equals(passwordText.getText())) {

loggedIn = true;

} else {

loggedIn = false;

}

credentials.close();

} catch (IOException e) {

e.getMessage();

}

return loggedIn;

}

/\*\*

\* Method for the user to return to the login screen

\*/

public static void returnToLogin() {

loginPanel.setVisible(true);

}

/\*\*

\* Accessor method to get the username of the user

\* @return a String within the userText text field

\*/

public static String getUsername() {

return userText.getText();

}

}

Register.java

**package** expenditureprogram;

**import** java.awt.BorderLayout;

**import** java.awt.Font;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**import** java.io.IOException;

**import** java.io.RandomAccessFile;

**import** javax.swing.JButton;

**import** javax.swing.JLabel;

**import** javax.swing.JOptionPane;

**import** javax.swing.JPanel;

**import** javax.swing.JPasswordField;

**import** javax.swing.JTextField;

**public** **class** Register **extends** JPanel {

**private** **static** **boolean** *isRegistered* = **false**;

// text field declaration

**private** **static** JTextField *userText*;

**private** **static** JPasswordField *passwordText*;

// button declaration

**private** **static** JButton *register*;

**private** **static** JButton *back*;

// label declaration

**private** **static** JLabel *userLabel*;

**private** **static** JLabel *passwordLabel*;

**private** **static** JLabel *title*;

// string declaration

**private** String username;

**private** String password;

// constructor

**public** Register() {

**super**(**new** BorderLayout());

// adds title to the middle of the panel

*title* = **new** JLabel("Account Registration");

*title*.setBounds(125, 10, 300, 25);

*title*.setFont(**new** Font ("Verdana", 1, 20));

add(*title*);

// sets title and location of username

*userLabel* = **new** JLabel("Create a username");

*userLabel*.setBounds(75, 80, 300, 25);

// adds the label onto the panel

add(*userLabel*);

// indicates the length of the text field

*userText* = **new** JTextField(20);

// the location of the text field

*userText*.setBounds(200,80,165,25);

add(*userText*);

// initializing password label

*passwordLabel* = **new** JLabel("Create a password");

*passwordLabel*.setBounds(75, 160, 300, 25);

add(*passwordLabel*);

// changes the password into dots, hides the password information

*passwordText* = **new** JPasswordField();

*passwordText*.setBounds(200, 160, 165, 25);

add(*passwordText*);

// register button

*register* = **new** JButton ("Register");

*register*.setBounds(270, 240, 100, 25);

add(*register*);

// once the register button is clicked, calls on the createAccount method

*register*.addActionListener (**new** ActionListener() {

**public** **void** actionPerformed (ActionEvent e) {

**if** (*userText*.getText().equals("") || *passwordText*.~~getText~~().equals("")) {

JOptionPane.*showMessageDialog*(**null**, "Username and password fields cannot be left blank. \nPlease try again.", "Login Error",

JOptionPane.***ERROR\_MESSAGE***);

} **else** {

createAccount();

**if** (*isRegistered* = **true**) {

~~hide~~();

Login.*returnToLogin*();

}

}

}

});

// back button to return to the login screen

*back* = **new** JButton ("Back to Login");

*back*.setBounds(75, 240, 125, 25);

add(*back*);

*back*.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed (ActionEvent e) {

~~hide~~();

Login.*returnToLogin*();

}

});

setLayout(**null**);

}

// method to create an account for the user

// account for error when nothing is entered and button is clicked

**public** **void** createAccount() {

// gets the user input in the username and password text fields

String username = *userText*.getText();

String password = *passwordText*.~~getText~~();

String modifiedPassword = "";

//String savedCredentials = "";

**try** {

// creates a new text file storing the user's credentials

RandomAccessFile credentials = **new** RandomAccessFile("users.txt", "rw");

**for** (**int** i = 0; i < password.length(); ++i) {

// encodes the password using the Caesar cipher by shifting characters of password back three letters

**int** k = password.charAt(i) - '\u0003';

modifiedPassword += (**char**)k;

}

credentials.writeBytes(username + "\\" + modifiedPassword + "\n");

credentials.writeBytes("");

} **catch** (IOException e) {

System.***out***.println("Error with saving file.");

*isRegistered* = **false**;

}

*userText*.setText("");

*passwordText*.setText("");

JOptionPane.*showMessageDialog*(**null**, "Account successfully created. You will be redirected to the login page.", "Account successfully created",

JOptionPane.***INFORMATION\_MESSAGE***);

*isRegistered* = **true**;

}

**public** **boolean** userRegistered() {

**return** *isRegistered*;

}

**public** String getName() {

**return** username;

}

**public** String getPassword() {

**return** password;

}

}

Menu.java

package expenditureprogram;

import java.awt.Font;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.File;

import java.io.IOException;

import java.util.Scanner;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JOptionPane;

import javax.swing.JPanel;

public class Menu extends JFrame {

// declaration of variables

private static JPanel menuPanel;

private JLabel menuTitle;

private JLabel userLabel;

private JButton exitButton;

private JButton existingSpreadsheetButton;

private JButton newSpreadsheetButton;

private static String spreadsheetName;

// constructor

public Menu() {

super("Expenditures Menu");

menuPanel = new JPanel();

this.setSize(500, 500);

// default close operation

this.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

this.setResizable(false);

// adds the JPanel to the window

this.add(menuPanel);

menuPanel.setLayout(null);

// initialization of labels and buttons

menuTitle = new JLabel("Expenditures Menu");

menuTitle.setBounds(145, 10, 400, 25);

menuTitle.setFont(new Font("Verdana", 1, 20));

String welcomeMessage = "Welcome " + Login.getUsername() + "!";

userLabel = new JLabel(welcomeMessage);

userLabel.setBounds(120, 65, 300, 25);

userLabel.setFont(new Font("Verdana", 1, 20));

existingSpreadsheetButton = new JButton("Select Existing Spreadsheet");

existingSpreadsheetButton.setBounds(25, 110, 200, 50);

existingSpreadsheetButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent arg0) {

// redirects user to new JFrame to choose saved spreadsheets

if (checkFileExists()) {

new SpreadsheetSelect();

dispose();

// there are no spreadsheets that are saved because the file containing the spreadsheet

// names does not exist

} else {

JOptionPane.showMessageDialog(null, "You have no saved spreadsheets!",

"Spreadsheet loading error", JOptionPane.INFORMATION\_MESSAGE);

}

}

});

newSpreadsheetButton = new JButton("Create New Spreadsheet");

newSpreadsheetButton.setBounds(250, 110, 200, 50);

newSpreadsheetButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent arg0) {

spreadsheetName = JOptionPane.showInputDialog("Enter a name for your spreadsheet:");

// the spreadsheet name cannot be left blank

if (spreadsheetName == null || (spreadsheetName != null && ("".equals(spreadsheetName)))) {

JOptionPane.showMessageDialog(null, "Spreadsheet name cannot be left blank.",

"Spreadsheet Name Error", JOptionPane.INFORMATION\_MESSAGE);

// the spreadsheet name cannot be the same name as a previously stored spreadsheet

} else if (checkNameExists(spreadsheetName)){

JOptionPane.showMessageDialog(null, "This spreadsheet name already exists! \nPlease choose a different name.",

"Spreadsheet Name Error", JOptionPane.ERROR\_MESSAGE);

} else {

Spreadsheet sheet = new Spreadsheet(spreadsheetName, new ExpenseLinkedList());

new DisplaySpreadsheet(sheet);

dispose();

}

}

});

exitButton = new JButton("Log out");

exitButton.setBounds(175, 250, 125, 40);

exitButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent arg0) {

// TODO Auto-generated method stub

System.exit(0);

}

});

// adding the buttons and labels to the menu panel

menuPanel.add(menuTitle);

menuPanel.add(userLabel);

menuPanel.add(existingSpreadsheetButton);

menuPanel.add(newSpreadsheetButton);

menuPanel.add(exitButton);

this.setVisible(true);

}

/\*\*

\* Checks whether the user input for the spreadsheet name is stored in listOfSpreadsheets.txt

\* @param name

\* @return true if the spreadsheet name already exists, false otherwise

\*/

public boolean checkNameExists(String name) {

try {

// file name is specified

File file = new File("listOfSpreadsheets.txt");

// create the file if the file is not already created

if (!file.exists()) {

file.createNewFile();

}

Scanner scanner = new Scanner(file);

while (scanner.hasNextLine()) {

String line = scanner.nextLine();

if (line.equals(name)) {

scanner.close();

return true;

}

}

scanner.close();

} catch (IOException e) {

e.printStackTrace();

}

return false;

}

/\*\*

\* Checks whether the listOfSpreadsheets.txt exists

\* Useful when the user logs in the first time and does not have any saved spreadsheets

\* @return true if it does, false otherwise

\*/

public boolean checkFileExists() {

// file name is specified

File file = new File("listOfSpreadsheets.txt");

// create the file if the file is not already created

if (!file.exists()) {

return false;

} else {

return true;

}

}

/\*\*

\* Returns the user input for the spreadsheet name

\* @return

\*/

public static String getSpreadsheetName() {

return spreadsheetName;

}

/\*\*

\* Method that returns the user to the main menu

\*/

public static void returnToMenu() {

menuPanel.setVisible(true);

}

}

Expense.java

**package** expenditureprogram;

**public** **class** Expense {

**private** String expenseName;

**private** **double** januaryExpenses;

**private** **double** februaryExpenses;

**private** **double** marchExpenses;

**private** **double** aprilExpenses;

**private** **double** mayExpenses;

**private** **double** juneExpenses;

**private** **double** julyExpenses;

**private** **double** augustExpenses;

**private** **double** septemberExpenses;

**private** **double** octoberExpenses;

**private** **double** novemberExpenses;

**private** **double** decemberExpenses;

//empty constructor

**public** Expense() {

}

**public** Expense(String newName, **double** jan, **double** feb, **double** mar, **double** apr, **double** may, **double** jun, **double** jul, **double** august, **double** sept, **double** oct, **double** nov, **double** dec) {

setExpenseName(newName);

}

**public** **void** setExpenseName(String newName) {

expenseName = newName;

}

**public** **void** setJanuary(**double** jan) {

januaryExpenses = jan;

}

**public** **void** setFebruary(**double** feb) {

februaryExpenses = feb;

}

**public** **void** setMarch(**double** mar) {

marchExpenses = mar;

}

**public** **void** setApril(**double** apr) {

aprilExpenses = apr;

}

**public** **void** setMay(**double** may) {

mayExpenses = may;

}

**public** **void** setJune(**double** jun) {

juneExpenses = jun;

}

**public** **void** setJuly(**double** jul) {

julyExpenses = jul;

}

**public** **void** setAugust(**double** aug) {

augustExpenses = aug;

}

**public** **void** setSeptember(**double** sept) {

septemberExpenses = sept;

}

**public** **void** setOctober(**double** oct) {

octoberExpenses = oct;

}

**public** **void** setNovember(**double** nov) {

novemberExpenses = nov;

}

**public** **void** setDecember(**double** dec) {

decemberExpenses = dec;

}

**public** String getExpenseName() {

**return** expenseName;

}

**public** **double** getJanuary() {

**return** januaryExpenses;

}

**public** **double** getFebruary() {

**return** februaryExpenses;

}

**public** **double** getMarch() {

**return** marchExpenses;

}

**public** **double** getApril() {

**return** aprilExpenses;

}

**public** **double** getMay() {

**return** mayExpenses;

}

**public** **double** getJune() {

**return** juneExpenses;

}

**public** **double** getJuly() {

**return** julyExpenses;

}

**public** **double** getAugust() {

**return** augustExpenses;

}

**public** **double** getSeptember() {

**return** septemberExpenses;

}

**public** **double** getOctober() {

**return** octoberExpenses;

}

**public** **double** getNovember() {

**return** novemberExpenses;

}

**public** **double** getDecember() {

**return** decemberExpenses;

}

**public** **double** getYearlyTotal() {

**return** januaryExpenses + februaryExpenses + marchExpenses + aprilExpenses + mayExpenses + juneExpenses + julyExpenses

+ augustExpenses + septemberExpenses + octoberExpenses + novemberExpenses + decemberExpenses;

}

}

ExpenseNode.java

**package** expenditureprogram;

**public** **class** ExpenseNode {

**private** ExpenseNode next;

**private** ExpenseNode prev;

**private** Expense exp;

**public** ExpenseNode(Expense exp, ExpenseNode next, ExpenseNode prev) {

**this**.exp = exp;

**this**.next = next;

**this**.prev = prev;

}

**public** **void** setValue (Expense exp) {

**this**.exp = exp;

}

**public** **void** setNext (ExpenseNode node) {

next = node;

}

**public** **void** setPrev (ExpenseNode node) {

prev = node;

}

**public** Expense getExpense() {

**return** exp;

}

**public** ExpenseNode getNext() {

**return** next;

}

**public** ExpenseNode getPrev() {

**return** prev;

}

}

ExpenseLinkedList.java

**package** expenditureprogram;

**public** **class** ExpenseLinkedList {

**private** ExpenseNode head;

**private** ExpenseNode tail;

**private** **int** size = 0;

/\*\*

\* Method returns the size of the linked list

\* **@return** the private variable size

\*/

**public** **int** getSize() {

**return** size;

}

/\*\*

\* Method to add a node at the end of the linked list

\* **@param** expense, the Expense must be a valid expense

\*/

**public** **void** addBack (Expense expense) {

**if** (head == **null**) {

head = **new** ExpenseNode(expense, **null**, **null**);

tail = head;

} **else** {

ExpenseNode node = **new** ExpenseNode (expense, **null**, tail);

**this**.tail.setNext(node);

**this**.tail = node;

}

size++;

}

/\*\*

\* Removes the head of the linked list

\* **@return** the local variable temp, which is the deleted ExpenseNode head

\*/

**public** ExpenseNode removeFront() {

// the linked list is already empty, so null is returned

**if** (head == **null**) {

**return** **null**;

}

ExpenseNode temp = head;

head = head.getNext();

// unlink it from the linked list

temp.setNext(**null**);

// size of linked list decreases by one

size--;

**return** temp;

}

**public** **void** deleteAt(**int** i) {

// if the linked list is empty, nothing happens

**if** (head == **null**) {

**return**;

}

// store the head node

ExpenseNode temp = head;

**if** (i == 0) {

head = temp.getNext();

**return**;

}

// search for the previous node of the node to be deleted

**for** (**int** j = 0; temp != **null** && j < i-1; j++) {

temp = temp.getNext();

}

// if the selected index is more than the number of nodes, nothing is returned

**if** (temp == **null** || temp.getNext() == **null**) {

**return**;

}

// the ExpenseNode temp.getNext is the selected node to be deleted

// store the pointer to the next of the ExpenseNode to be deleted

ExpenseNode next = temp.getNext().getNext();

// unlink the deleted node from the list

temp.setNext(next);

// size of the linked list decrements by one

size--;

}

// method to return a Expense at a specific index

**public** Expense getExpense(**int** i) {

ExpenseNode current = head;

// current index of node

**int** count = 0;

**while** (current != **null**) {

// return the expense if the index matches

**if** (count == i) {

**return** current.getExpense();

}

count++;

current = current.getNext();

}

// the user was asking for a non-existent element, so

// assert fail

**assert** (**false**);

**return** **null**;

}

}

Spreadsheet.java

**package** expenditureprogram;

**import** javax.swing.table.AbstractTableModel;

**public** **class** Spreadsheet **extends** AbstractTableModel {

// testing with column names and cells

**private** String[] months = {"Expense Name", "Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sept", "Oct", "Nov", "Dec",

"Yearly Total" };

**private** ExpenseLinkedList list;

**private** String sheetName;

/\*\*

\* Constructor

\*/

**public** Spreadsheet(String name, ExpenseLinkedList list) {

sheetName = name;

**this**.list = list;

}

/\*\*

\* Accessor method to return the spreadsheet name

\* **@return** instance variable sheetName

\*/

**public** String getSpreadsheetName() {

**return** sheetName;

}

/\*\*

\* Mutator method to change the spreadsheet name

\* **@param** name, a new name for the spreadsheet must be declared

\*/

**public** **void** setSpreadsheetName(String name) {

sheetName = name;

}

/\*\*

\* Method to remove an expense from the spreadsheet

\* **@param** i, the index of the row must be specified

\*/

**public** **void** removeExpense(**int** i) {

// if the

**if** (i == 0) {

list.removeFront();

} **else** {

list.deleteAt(i);

}

fireTableRowsDeleted(i, i);

fireTableDataChanged();

}

/\*\*

\* Method to add an expense to the spreadsheet

\* **@param** expense, a valid Expense must be entered

\*/

**public** **void** addExpense(Expense expense) {

**int** row = list.getSize();

list.addBack(expense);

fireTableRowsInserted(row, row);

}

/\*\*

\* Returns the number of columns in the spreadsheet

\*/

**public** **int** getColumnCount() {

**return** months.length;

}

/\*\*

\* Return the number of rows in the spreadsheet

\*/

**public** **int** getRowCount() {

**return** list.getSize();

}

/\*\*

\* Return the names of each column in the spreadsheet

\*

\*/

**public** String getColumnName(**int** col) {

**return** months[col];

}

/\*\*

\* Change the value of a cell in an expense on the spreadsheet

\*/

**public** **void** setValueAt(Object value, **int** rowIndex, **int** colIndex) {

Expense expense = list.getExpense(rowIndex);

String strValue = String.*valueOf*(value);

// change this value to the original cell value

**double** doubleValue = 0;

**if** (colIndex == 0) {

expense.setExpenseName(strValue);

}

**try** {

doubleValue = Double.*valueOf*(strValue);

} **catch** (NumberFormatException e) {

e.getMessage();

}

**switch** (colIndex) {

**case** 1:

expense.setJanuary(doubleValue);

**case** 2:

expense.setFebruary(doubleValue);

**case** 3:

expense.setMarch(doubleValue);

**case** 4:

expense.setApril(doubleValue);

**case** 5:

expense.setMay(doubleValue);

**case** 6:

expense.setJune(doubleValue);

**case** 7:

expense.setJuly(doubleValue);

**case** 8:

expense.setAugust(doubleValue);

**case** 9:

expense.setSeptember(doubleValue);

**case** 10:

expense.setOctober(doubleValue);

**case** 11:

expense.setNovember(doubleValue);

**case** 12:

expense.setDecember(doubleValue);

}

fireTableRowsUpdated(rowIndex, colIndex);

}

/\*\*

\* Return the object of a cell in the expense

\*/

**public** Object getValueAt(**int** rowIndex, **int** colIndex) {

Expense expense = list.getExpense(rowIndex);

**if** (expense == **null**) {

**return** **null**;

}

**switch** (colIndex) {

**case** 0:

**return** expense.getExpenseName();

**case** 1:

**return** expense.getJanuary();

**case** 2:

**return** expense.getFebruary();

**case** 3:

**return** expense.getMarch();

**case** 4:

**return** expense.getApril();

**case** 5:

**return** expense.getMay();

**case** 6:

**return** expense.getJune();

**case** 7:

**return** expense.getJuly();

**case** 8:

**return** expense.getAugust();

**case** 9:

**return** expense.getSeptember();

**case** 10:

**return** expense.getOctober();

**case** 11:

**return** expense.getNovember();

**case** 12:

**return** expense.getDecember();

**case** 13:

**return** expense.getYearlyTotal();

**default**:

**return** **null**;

}

}

}

DisplaySpreadsheet.java

package expenditureprogram;

import java.awt.BorderLayout;

import java.awt.Dimension;

import java.awt.Font;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.print.PrinterException;

import java.io.BufferedWriter;

import java.io.File;

import java.io.FileWriter;

import java.io.IOException;

import java.io.PrintWriter;

import java.text.MessageFormat;

import java.util.Scanner;

import java.util.logging.Level;

import java.util.logging.Logger;

import javax.swing.BoxLayout;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JOptionPane;

import javax.swing.JPanel;

import javax.swing.JScrollPane;

import javax.swing.JTable;

import javax.swing.JTextField;

import javax.swing.ListSelectionModel;

import javax.swing.table.JTableHeader;

import javax.swing.table.TableModel;

public class DisplaySpreadsheet extends JFrame {

// declaration and initialization of variables

// dimensions of the window

private final static int WIDTH = 1180;

private final static int HEIGHT = 500;

private static Spreadsheet sheet;

private JLabel spreadsheetNameLabel;

private JButton addRow;

private JButton deleteExpense;

private JButton editExpense;

private JButton back;

private JButton save;

private JButton print;

private JButton getMonthlyTotals;

private JTextField janField;

private JTextField febField;

private JTextField marField;

private JTextField aprField;

private JTextField mayField;

private JTextField junField;

private JTextField julField;

private JTextField augField;

private JTextField septField;

private JTextField octField;

private JTextField novField;

private JTextField decField;

private JTextField yearField;

private static JPanel mainPanel;

private static JPanel topPanel;

private static JPanel centerPanel;

private static JPanel bottomPanel;

private JTable spreadsheet;

private static int rowIndex;

private static boolean isEditedClicked;

private boolean isSpreadsheetSaved = false;

private static String spreadsheetName;

// default empty constructor

public DisplaySpreadsheet() {

}

public DisplaySpreadsheet(Spreadsheet sheet) {

// initializing the name and size of the JFrame

super("Expenditures");

setPreferredSize(new Dimension(WIDTH, HEIGHT));

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

// declaration of panels

mainPanel = new JPanel();

topPanel = new JPanel();

centerPanel = new JPanel();

bottomPanel = new JPanel();

// setting the layout and adding the panels to the main panel

mainPanel.setLayout(new BoxLayout(mainPanel, BoxLayout.Y\_AXIS));

mainPanel.add(topPanel, BorderLayout.NORTH);

mainPanel.add(centerPanel, BorderLayout.CENTER);

mainPanel.add(bottomPanel, BorderLayout.SOUTH);

// get the name of the spreadsheet

spreadsheetName = sheet.getSpreadsheetName();

// initialize the spreadsheet that is being displayed

DisplaySpreadsheet.sheet = sheet;

// initializing the JTable

spreadsheet = new JTable(DisplaySpreadsheet.sheet);

spreadsheet.setPreferredScrollableViewportSize(new Dimension(WIDTH, HEIGHT));

spreadsheet.setFillsViewportHeight(true);

// prevents user from dragging columns around

spreadsheet.getTableHeader().setReorderingAllowed(false);

// user can only select one row at a time

spreadsheet.setSelectionMode(ListSelectionModel.SINGLE\_SELECTION);

// gets the column names of the spreadsheet

JTableHeader header = spreadsheet.getTableHeader();

// adding the buttons, labels, and text fields of the panel

addRow = new JButton("Add an expense.");

addRow.setBounds(75, 25, 150, 30);

addRow.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent arg0) {

// user chooses to add an expense, a new JFrame will pop up to add this expense

EditExpense edit = new EditExpense();

edit.setVisible(true);

edit.setLocationRelativeTo(null);

edit.setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);

// the spreadsheet will change and therefore it is not saved

isSpreadsheetSaved = false;

}

});

editExpense = new JButton("Edit an expense.");

editExpense.setBounds(95, 25, 150, 30);

editExpense.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent arg0) {

// the user chooses to edit an existing expense

isEditedClicked = true;

// determines whether the spreadsheet is empty or not, if so, a message will indicate that it is empty

if (sheet.getRowCount() == 0) {

JOptionPane.showMessageDialog(null, "There are no expenses in the spreadsheet!",

"Spreadsheet is empty", JOptionPane.ERROR\_MESSAGE);

// otherwise, the spreadsheet is not empty and a new JFrame will open up

// the text fields will display the current values for the expenses

} else if (spreadsheet.getSelectedRow() >= 0){

rowIndex = spreadsheet.getSelectedRow();

EditExpense editedExpense = new EditExpense();

String expenseName = spreadsheet.getValueAt(rowIndex, 0).toString();

String janExpense = spreadsheet.getValueAt(rowIndex, 1).toString();

String febExpense = spreadsheet.getValueAt(rowIndex, 2).toString();

String marExpense = spreadsheet.getValueAt(rowIndex, 3).toString();

String aprExpense = spreadsheet.getValueAt(rowIndex, 4).toString();

String mayExpense = spreadsheet.getValueAt(rowIndex, 5).toString();

String junExpense = spreadsheet.getValueAt(rowIndex, 6).toString();

String julExpense = spreadsheet.getValueAt(rowIndex, 7).toString();

String augExpense = spreadsheet.getValueAt(rowIndex, 8).toString();

String septExpense = spreadsheet.getValueAt(rowIndex, 9).toString();

String octExpense = spreadsheet.getValueAt(rowIndex, 10).toString();

String novExpense = spreadsheet.getValueAt(rowIndex, 11).toString();

String decExpense = spreadsheet.getValueAt(rowIndex, 12).toString();

editedExpense.expenseName.setText(expenseName);

editedExpense.expenseJan.setText(janExpense);

editedExpense.expenseFeb.setText(febExpense);

editedExpense.expenseMar.setText(marExpense);

editedExpense.expenseApr.setText(aprExpense);

editedExpense.expenseMay.setText(mayExpense);

editedExpense.expenseJun.setText(junExpense);

editedExpense.expenseJul.setText(julExpense);

editedExpense.expenseAug.setText(augExpense);

editedExpense.expenseSept.setText(septExpense);

editedExpense.expenseOct.setText(octExpense);

editedExpense.expenseNov.setText(novExpense);

editedExpense.expenseDec.setText(decExpense);

isSpreadsheetSaved = false;

editedExpense.setVisible(true);

editedExpense.setLocationRelativeTo(null);

editedExpense.setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);

} else {

JOptionPane.showMessageDialog(null,

"Please select an expense to edit.", "Select an expense.",

JOptionPane.ERROR\_MESSAGE);

}

}

});

deleteExpense = new JButton("Delete an Expense.");

deleteExpense.setBounds(400, 25, 150, 30);

deleteExpense.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent arg0) {

rowIndex = spreadsheet.getSelectedRow();

// checks if the expense exists in the JTable

if (rowIndex >= 0) {

// calls the method that deletes the row at that specific index

deleteRowAtIndex(rowIndex);

isSpreadsheetSaved = false;

// otherwise, the spreadsheet is empty and nothing is deleted

} else if (sheet.getRowCount() == 0) {

JOptionPane.showMessageDialog(null,

"The spreadsheet is empty. There are no more expenses to delete!", "Spreadsheet is empty",

JOptionPane.ERROR\_MESSAGE);

}

else {

JOptionPane.showMessageDialog(null,

"Please select an expense to delete.", "Select an expense.",

JOptionPane.ERROR\_MESSAGE);

}

}

});

save = new JButton("Save spreadsheet.");

save.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent arg0) {

// save the contents of the spreadsheet/JTable into a text file

saveSheet();

// save the spreadsheet name in a text file

storeSpreadsheetNames();

}

});

print = new JButton("Print");

print.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent arg0) {

print();

}

});

back = new JButton("Back to Menu.");

back.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent arg0) {

// if the spreadsheet is not saved, the user is prompted to reconsider this decision

// acts as a reminder for them to save their spreadsheet unless otherwise indicated

if (isSpreadsheetSaved == false) {

int reply = JOptionPane.showConfirmDialog(null,

"Your spreadsheet has not been saved. Are you sure you want to leave?",

"Spreadsheet not saved", JOptionPane.YES\_NO\_OPTION);

if (reply == JOptionPane.YES\_OPTION) {

// user indicates that the spreadsheet will not be saved, redirected back to the menu

new Menu();

dispose();

// user chooses to reconsider, nothing happens to the spreadsheet

} else {

return;

}

// the spreadsheet is confirmed to be saved, and the user can go back to the menu

} else {

new Menu();

dispose();

}

}

});

getMonthlyTotals = new JButton("Get Monthly Totals");

getMonthlyTotals.setPreferredSize(new Dimension(150, 20));

getMonthlyTotals.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent arg0) {

// calls the method that sets the text fields to display the monthly/yearly totals of all expenses

getColumnsTotals();

}

});

spreadsheetNameLabel = new JLabel(spreadsheetName);

spreadsheetNameLabel.setBounds(65, 10, 400, 25);

spreadsheetNameLabel.setFont(new Font("Verdana", 1, 20));

janField = new JTextField(5);

febField = new JTextField(5);

marField = new JTextField(5);

aprField = new JTextField(5);

mayField = new JTextField(5);

junField = new JTextField(5);

julField = new JTextField(5);

augField = new JTextField(5);

septField = new JTextField(5);

octField = new JTextField(5);

novField = new JTextField(5);

decField = new JTextField(5);

yearField = new JTextField(5);

janField.setText("Jan");

febField.setText("Feb");

marField.setText("Mar");

aprField.setText("Apr");

mayField.setText("May");

junField.setText("Jun");

julField.setText("Jul");

augField.setText("Aug");

septField.setText("Sept");

octField.setText("Oct");

novField.setText("Nov");

decField.setText("Dec");

yearField.setText("Year");

// adding the buttons, text fields, labels and JTable to respective panels

topPanel.add(spreadsheetNameLabel, BorderLayout.NORTH);

topPanel.add(save);

topPanel.add(addRow);

topPanel.add(deleteExpense);

topPanel.add(editExpense);

topPanel.add(print);

topPanel.add(back);

centerPanel.add(getMonthlyTotals);

centerPanel.add(janField);

centerPanel.add(febField);

centerPanel.add(marField);

centerPanel.add(aprField);

centerPanel.add(mayField);

centerPanel.add(junField);

centerPanel.add(julField);

centerPanel.add(augField);

centerPanel.add(septField);

centerPanel.add(octField);

centerPanel.add(novField);

centerPanel.add(decField);

centerPanel.add(yearField);

bottomPanel.add(header, BorderLayout.CENTER);

bottomPanel.add(spreadsheet, BorderLayout.CENTER);

// adds a scroll pane for the user

JScrollPane scrollPane = new JScrollPane(spreadsheet, JScrollPane.VERTICAL\_SCROLLBAR\_AS\_NEEDED, JScrollPane.HORIZONTAL\_SCROLLBAR\_AS\_NEEDED);

scrollPane.setVisible(true);

mainPanel.add(scrollPane, BorderLayout.CENTER);

// sets the content pane of the window to be the main panel that consists of all other panels and features

setContentPane(mainPanel);

pack();

// displays the window

setVisible(true);

}

/\*\*

\* Method that sums up all the monthly and yearly expenses

\* Prints out the sum as a string in the text field

\*/

public void getColumnsTotals() {

int rowCount = spreadsheet.getRowCount();

double janTotal = 0;

double febTotal = 0;

double marTotal = 0;

double aprTotal = 0;

double mayTotal = 0;

double junTotal = 0;

double julTotal = 0;

double augTotal = 0;

double septTotal = 0;

double octTotal = 0;

double novTotal = 0;

double decTotal = 0;

double yearlyTotal = 0;

for (int i = 0; i < rowCount; i++) {

janTotal += Double.parseDouble(spreadsheet.getValueAt(i, 1).toString());

febTotal += Double.parseDouble(spreadsheet.getValueAt(i, 2).toString());

marTotal += Double.parseDouble(spreadsheet.getValueAt(i, 3).toString());

aprTotal += Double.parseDouble(spreadsheet.getValueAt(i, 4).toString());

mayTotal += Double.parseDouble(spreadsheet.getValueAt(i, 5).toString());

junTotal += Double.parseDouble(spreadsheet.getValueAt(i, 6).toString());

julTotal += Double.parseDouble(spreadsheet.getValueAt(i, 7).toString());

augTotal += Double.parseDouble(spreadsheet.getValueAt(i, 8).toString());

septTotal += Double.parseDouble(spreadsheet.getValueAt(i, 9).toString());

octTotal += Double.parseDouble(spreadsheet.getValueAt(i, 10).toString());

novTotal += Double.parseDouble(spreadsheet.getValueAt(i, 11).toString());

decTotal += Double.parseDouble(spreadsheet.getValueAt(i, 12).toString());

yearlyTotal += Double.parseDouble(spreadsheet.getValueAt(i, 13).toString());

}

janField.setText(String.valueOf(janTotal));

febField.setText(String.valueOf(febTotal));

marField.setText(String.valueOf(marTotal));

aprField.setText(String.valueOf(aprTotal));

mayField.setText(String.valueOf(mayTotal));

junField.setText(String.valueOf(junTotal));

julField.setText(String.valueOf(julTotal));

augField.setText(String.valueOf(augTotal));

septField.setText(String.valueOf(septTotal));

octField.setText(String.valueOf(octTotal));

novField.setText(String.valueOf(novTotal));

decField.setText(String.valueOf(decTotal));

yearField.setText(String.valueOf(yearlyTotal));

}

/\*\*

\* Adds a new expense to the spreadsheet and JTable

\* @param newExpense must be declared

\*/

public static void addRowToSpreadsheet(Expense newExpense) {

sheet.addExpense(newExpense);

}

/\*\*

\* Deletes an expense from the spreadsheet at a specific index

\* @param i, the index at which the expense is deleted must be specified

\*/

public static void deleteRowAtIndex(int i) {

sheet.removeExpense(i);

}

/\*\*

\* Method that retrieves the row that the user clicked on in the JTable

\* @return the index of the row that the user selected

\*/

public static int getRowIndex() {

return rowIndex;

}

/\*\*

\* Mutator method that sets whether an expense is edited or not

\* @param reset, sets isEditedClicked to true or false depending on whether the expense is edited

\*/

public static void setEditedClicked(boolean reset) {

isEditedClicked = reset;

}

/\*\*

\* Accessor method that returns the boolean variable isEditedClicked

\* @return the variable isEditedClicked

\*/

public static boolean getEditedClicked() {

return isEditedClicked;

}

/\*\*

\* Accessor method that returns the name of the spreadsheet

\* @return the variable spreadsheetName

\*/

public String getSpreadsheetName() {

return spreadsheetName;

}

/\*\*

\* Accessor method that returns the spreadsheet

\* @return the variable sheet

\*/

public static Spreadsheet getSpreadsheet() {

return sheet;

}

/\*\*

\* Method that saves the spreadsheet as an Excel file and a text file

\*/

public void saveSheet() {

// saving a copy as an Excel file

boolean ifSaved = false;

try {

File tableInExcel = new File(spreadsheetName + ".xls");

TableModel model = spreadsheet.getModel();

FileWriter excel = new FileWriter(tableInExcel);

// writes the column names into Excel

for (int i = 0; i < model.getColumnCount(); i++) {

excel.write(model.getColumnName(i) + "\t");

}

excel.write("\n");

// writes a new expense on a new line

for (int i = 0; i < model.getColumnCount(); i++) {

for (int j = 0; j < model.getColumnCount(); j++) {

String data = String.valueOf(model.getValueAt(i, j));

if (data == "null") {

data = "";

}

excel.write(data + "\t");

}

excel.write("\n");

}

excel.close();

ifSaved = true;

} catch (IOException e) {

Logger.getLogger(DisplaySpreadsheet.class.getName()).log(Level.SEVERE, null, e);

}

// saving a copy as text file, easier to load the spreadsheet

String filePath = spreadsheetName + ".txt";

File file = new File(filePath);

try {

FileWriter fw = new FileWriter(file);

BufferedWriter bw = new BufferedWriter(fw);

for (int i = 0; i < spreadsheet.getRowCount(); i++) {

for (int j = 0; j < spreadsheet.getColumnCount(); j++) {

bw.write(String.valueOf(spreadsheet.getValueAt(i, j)));

// separate each entry with an asterisk

bw.write("\*");

}

// new line for a new expense

bw.newLine();

}

bw.close();

fw.close();

ifSaved = true;

} catch (IOException e) {

Logger.getLogger(DisplaySpreadsheet.class.getName()).log(Level.SEVERE, null, e);

}

// once the files are saved, display a message to the user that the files are saved

if (ifSaved) {

JOptionPane.showMessageDialog(null, "File Successfully Saved!", "File Saved.",

JOptionPane.INFORMATION\_MESSAGE);

isSpreadsheetSaved = true;

}

}

/\*\*

\* Method that stores the name of the spreadsheet being used

\*/

public static void storeSpreadsheetNames() {

boolean alreadyExists = false;

try {

// file name is specified

File file = new File ("listOfSpreadsheets.txt");

// create the file if the file is not already created

if (!file.exists()) {

file.createNewFile();

}

Scanner scanner = new Scanner (file);

while (scanner.hasNextLine()) {

String line = scanner.nextLine();

// if the spreadsheet name already exists in the file, this text file is not changed

if (line.equals(spreadsheetName)) {

alreadyExists = true;

}

}

scanner.close();

if (!alreadyExists) {

// if the spreadsheet name does not exist, add the name of the spreadsheet to the text file

FileWriter fw = new FileWriter(file,true);

BufferedWriter bw = new BufferedWriter(fw);

PrintWriter pw = new PrintWriter(bw);

pw.println(spreadsheetName);

pw.close();

}

} catch (IOException e) {

Logger.getLogger(DisplaySpreadsheet.class.getName()).log(Level.SEVERE, null, e);

}

}

private void print() {

MessageFormat header = new MessageFormat("Print Expenses");

try {

boolean complete = spreadsheet.print(JTable.PrintMode.NORMAL, header, null);

if (complete) {

JOptionPane.showMessageDialog(null, "Printing successful!");

}

} catch (PrinterException e) {

JOptionPane.showMessageDialog(null, "Could not print.", "Error", JOptionPane.INFORMATION\_MESSAGE);

}

}

public static void main(String[] args) {

// TODO Auto-generated method stub

new DisplaySpreadsheet();

}

}

SelectSpreadsheet.java

package expenditureprogram;

import java.awt.EventQueue;

import java.awt.Font;

import java.awt.GridBagConstraints;

import java.awt.GridBagLayout;

import java.awt.Insets;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileReader;

import java.io.IOException;

import java.util.logging.Level;

import java.util.logging.Logger;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JPanel;

import javax.swing.JScrollPane;

public class SpreadsheetSelect extends JFrame {

private JButton back;

private JLabel existingLabel;

private static JPanel mainPanel;

private String [] array;

public SpreadsheetSelect() {

// sets the title of the JFrame

super("Existing Spreadsheets.");

// sets the size of the window

setSize(500, 500);

// default close operation

setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);

// the window cannot be resized

setResizable(false);

mainPanel = new JPanel();

mainPanel.setLayout(new GridBagLayout());

back = new JButton ("Back to Menu.");

//back.setBounds(350, 10, 125, 25);

back.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent arg0) {

// if the back button is pressed, user directed back to the menu and this frame closes

dispose();

new Menu();

}

});

// adds the labels and buttons to the panel

initJComponents();

// adds the vertical scroll pane to the panel and window

JScrollPane scrollPane = new JScrollPane(mainPanel, JScrollPane.VERTICAL\_SCROLLBAR\_AS\_NEEDED,

JScrollPane.HORIZONTAL\_SCROLLBAR\_NEVER);

this.add(scrollPane);

// sets the JFrame to be visible

setVisible(true);

}

/\*\*

\* Method that initializes all buttons and their layout and adds them to the main panel

\*/

public void initJComponents() {

// initializing the GridBagConstraints and adding components to it

GridBagConstraints gbc = new GridBagConstraints();

gbc.anchor = GridBagConstraints.NORTH;

gbc.fill = GridBagConstraints.VERTICAL;

gbc.gridx = 1;

gbc.gridy = 0;

// adding the label to the GridBagConstraint and JPanel

existingLabel = new JLabel ("Existing Spreadsheets");

existingLabel.setFont(new Font("Verdana", 1, 20));

mainPanel.add(existingLabel, gbc);

gbc.insets = new Insets(10,10,10,10);

gbc.gridx--;

// adding the back button to the panel

mainPanel.add(back, gbc);

// calls the method that parses the text file where all spreadsheet names are stored

// and initializes a String array to contain the spreadsheet names

addButtonsSpreadsheets();

gbc.gridwidth = 2;

for(int i = 0; i < array.length; i++) {

// add a new button for each spreadsheet name

JButton button = new JButton (array[i]);

int temp = i;

button.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent arg0) {

// when button is clicked, load the spreadsheet and dispose the window

loadSpreadsheet(array[temp]);

dispose();

}

});

gbc.gridy++;

// adds the buttons to the screen

mainPanel.add(button, gbc);

}

}

/\*\*

\* Method that stores the spreadsheet names from the text file into a String array

\* The array should be initialized after this method call

\*/

private void addButtonsSpreadsheets() {

// local variable that counts the number of lines in the listOfSpreadsheets text file

int counter = 0;

File spreadsheetNames = new File ("listOfSpreadsheets.txt");

try {

FileReader fr = new FileReader(spreadsheetNames);

BufferedReader br = new BufferedReader(fr);

String line = br.readLine();

while (line != null) {

// determines the number of spreadsheet names in the text file

counter++;

line = br.readLine();

}

br.close();

} catch (IOException e) {

e.printStackTrace();

}

// creates new array to store the spreadsheet names in the program

array = new String[counter];

// reset the counter to initialize the contents in the array

counter = 0;

// reads the file again

File spreadsheetNames2 = new File ("listOfSpreadsheets.txt");

try {

FileReader fr = new FileReader(spreadsheetNames2);

BufferedReader br = new BufferedReader(fr);

String line = br.readLine();

while (line != null) {

// store the spreadsheet names into an array

array[counter] = line;

counter++;

line = br.readLine();

}

br.close();

} catch (IOException e) {

e.printStackTrace();

}

}

/\*\*

\* Loads the spreadsheet that the user chooses to click

\* @param sheetName, takes in the name of the spreadsheet

\* On button click, the saved spreadsheet will be displayed

\*/

private void loadSpreadsheet(String sheetName) {

// creates a new spreadsheet displaying the saved information

Spreadsheet sheet = new Spreadsheet(sheetName, new ExpenseLinkedList());

new DisplaySpreadsheet(sheet);

String filePath = sheetName + ".txt";

File file = new File(filePath);

try {

FileReader fr = new FileReader(file);

BufferedReader br = new BufferedReader(fr);

Object [] lines = br.lines().toArray();

for (int i = 0; i < lines.length; i++) {

String [] parts = lines[i].toString().split("\\\*");

Expense loadExpense = new Expense();

loadExpense.setExpenseName(parts[0]);

for (int j = 1; j < parts.length; j++) {

switch (j) {

case 1:

loadExpense.setJanuary(Double.parseDouble(parts[j]));

case 2:

loadExpense.setFebruary(Double.parseDouble(parts[j]));

case 3:

loadExpense.setMarch(Double.parseDouble(parts[j]));

case 4:

loadExpense.setApril(Double.parseDouble(parts[j]));

case 5:

loadExpense.setMay(Double.parseDouble(parts[j]));

case 6:

loadExpense.setJune(Double.parseDouble(parts[j]));

case 7:

loadExpense.setJuly(Double.parseDouble(parts[j]));

case 8:

loadExpense.setAugust(Double.parseDouble(parts[j]));

case 9:

loadExpense.setSeptember(Double.parseDouble(parts[j]));

case 10:

loadExpense.setOctober(Double.parseDouble(parts[j]));

case 11:

loadExpense.setNovember(Double.parseDouble(parts[j]));

case 12:

loadExpense.setDecember(Double.parseDouble(parts[j]));

}

}

sheet.addExpense(loadExpense);

}

br.close();

} catch (IOException ex) {

Logger.getLogger(DisplaySpreadsheet.class.getName()).log(Level.SEVERE, null, ex);

}

}

public static void main(String[] args) {

// TODO Auto-generated method stub

EventQueue.invokeLater(new Runnable() {

public void run() {

try {

SpreadsheetSelect frame = new SpreadsheetSelect();

frame.setVisible(true);

} catch (Exception e) {

e.printStackTrace();

}

}

});

}

}

EditExpense.java

package expenditureprogram;

import java.awt.Font;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JOptionPane;

import javax.swing.JTextField;

public class EditExpense extends JFrame {

// declaration of variables

JTextField expenseName;

JTextField expenseJan;

JTextField expenseFeb;

JTextField expenseMar;

JTextField expenseApr;

JTextField expenseMay;

JTextField expenseJun;

JTextField expenseJul;

JTextField expenseAug;

JTextField expenseSept;

JTextField expenseOct;

JTextField expenseNov;

JTextField expenseDec;

JLabel panelTitle;

JLabel expenseNameLabel;

JLabel expenseJanLabel;

JLabel expenseFebLabel;

JLabel expenseMarLabel;

JLabel expenseAprLabel;

JLabel expenseMayLabel;

JLabel expenseJunLabel;

JLabel expenseJulLabel;

JLabel expenseAugLabel;

JLabel expenseSeptLabel;

JLabel expenseOctLabel;

JLabel expenseNovLabel;

JLabel expenseDecLabel;

JButton addExpense;

JButton exitButton;

Expense expense = new Expense();

public EditExpense() {

setTitle("Edit an expense");

setLayout(null);

setSize(500, 500);

setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);

// adding labels

panelTitle = new JLabel("Edit an Expense");

panelTitle.setBounds(150, 10, 300, 25);

panelTitle.setFont(new Font("Verdana", 1, 20));

add(panelTitle);

expenseNameLabel = new JLabel("Name of Expense");

expenseNameLabel.setBounds(75, 50, 300, 25);

add(expenseNameLabel);

expenseJanLabel = new JLabel("January Expenses");

expenseJanLabel.setBounds(75, 80, 300, 25);

add(expenseJanLabel);

expenseFebLabel = new JLabel("February Expenses");

expenseFebLabel.setBounds(75, 110, 300, 25);

add(expenseFebLabel);

expenseMarLabel = new JLabel("March Expenses");

expenseMarLabel.setBounds(75, 140, 300, 25);

add(expenseMarLabel);

expenseAprLabel = new JLabel("April Expenses");

expenseAprLabel.setBounds(75, 170, 300, 25);

add(expenseAprLabel);

expenseMayLabel = new JLabel("May Expenses");

expenseMayLabel.setBounds(75, 200, 300, 25);

add(expenseMayLabel);

expenseJunLabel = new JLabel("June Expenses");

expenseJunLabel.setBounds(75, 230, 300, 25);

add(expenseJunLabel);

expenseJulLabel = new JLabel("July Expenses");

expenseJulLabel.setBounds(75, 260, 300, 25);

add(expenseJulLabel);

expenseAugLabel = new JLabel("August Expenses");

expenseAugLabel.setBounds(75, 290, 300, 25);

add(expenseAugLabel);

expenseSeptLabel = new JLabel("September Expenses");

expenseSeptLabel.setBounds(75, 320, 300, 25);

add(expenseSeptLabel);

expenseOctLabel = new JLabel("October Expenses");

expenseOctLabel.setBounds(75, 350, 300, 25);

add(expenseOctLabel);

expenseNovLabel = new JLabel("November Expenses");

expenseNovLabel.setBounds(75, 380, 300, 25);

add(expenseNovLabel);

expenseDecLabel = new JLabel("December Expenses");

expenseDecLabel.setBounds(75, 410, 300, 25);

add(expenseDecLabel);

// adding text fields

// indicates the length of the text field

expenseName = new JTextField(20);

// the location of the text field

expenseName.setBounds(225, 50, 165, 25);

add(expenseName);

// indicates the length of the text field

expenseJan = new JTextField(20);

// the location of the text field

expenseJan.setBounds(225, 80, 165, 25);

add(expenseJan);

// indicates the length of the text field

expenseFeb = new JTextField(20);

// the location of the text field

expenseFeb.setBounds(225, 110, 165, 25);

add(expenseFeb);

// indicates the length of the text field

expenseMar = new JTextField(20);

// the location of the text field

expenseMar.setBounds(225, 140, 165, 25);

add(expenseMar);

// indicates the length of the text field

expenseApr = new JTextField(20);

// the location of the text field

expenseApr.setBounds(225, 170, 165, 25);

add(expenseApr);

// indicates the length of the text field

expenseMay = new JTextField(20);

// the location of the text field

expenseMay.setBounds(225, 200, 165, 25);

add(expenseMay);

// indicates the length of the text field

expenseJun = new JTextField(20);

// the location of the text field

expenseJun.setBounds(225, 230, 165, 25);

add(expenseJun);

// indicates the length of the text field

expenseJul = new JTextField(20);

// the location of the text field

expenseJul.setBounds(225, 260, 165, 25);

add(expenseJul);

// indicates the length of the text field

expenseAug = new JTextField(20);

// the location of the text field

expenseAug.setBounds(225, 290, 165, 25);

add(expenseAug);

// indicates the length of the text field

expenseSept = new JTextField(20);

// the location of the text field

expenseSept.setBounds(225, 320, 165, 25);

add(expenseSept);

// indicates the length of the text field

expenseOct = new JTextField(20);

// the location of the text field

expenseOct.setBounds(225, 350, 165, 25);

add(expenseOct);

// indicates the length of the text field

expenseNov = new JTextField(20);

// the location of the text field

expenseNov.setBounds(225, 380, 165, 25);

add(expenseNov);

// indicates the length of the text field

expenseDec = new JTextField(20);

// the location of the text field

expenseDec.setBounds(225, 410, 165, 25);

add(expenseDec);

addExpense = new JButton("Update Table");

addExpense.setBounds(75, 440, 125, 25);

addExpense.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent arg0) {

boolean isExpenseValid = true;

boolean isEditedClicked = DisplaySpreadsheet.getEditedClicked();

// checks if user input a name for their expense

if (expenseName.getText().equals("")) {

JOptionPane.showMessageDialog(null, "Expense Name cannot be left blank. \nPlease try again.",

"Input Error", JOptionPane.ERROR\_MESSAGE);

isExpenseValid = false;

} else if (isEditedClicked == false){

expense.setExpenseName(expenseName.getText());

} else {

DisplaySpreadsheet.getSpreadsheet().setValueAt(String.valueOf(expenseName.getText()), DisplaySpreadsheet.getRowIndex(), 0);

}

try {

// checks if user input is a positive number

if (Double.parseDouble(expenseJan.getText()) < 0 || Double.parseDouble(expenseFeb.getText()) < 0

|| Double.parseDouble(expenseJan.getText()) < 0

|| Double.parseDouble(expenseFeb.getText()) < 0

|| Double.parseDouble(expenseMar.getText()) < 0

|| Double.parseDouble(expenseApr.getText()) < 0

|| Double.parseDouble(expenseMay.getText()) < 0

|| Double.parseDouble(expenseJun.getText()) < 0

|| Double.parseDouble(expenseJul.getText()) < 0

|| Double.parseDouble(expenseAug.getText()) < 0

|| Double.parseDouble(expenseSept.getText()) < 0

|| Double.parseDouble(expenseOct.getText()) < 0

|| Double.parseDouble(expenseNov.getText()) < 0

|| Double.parseDouble(expenseDec.getText()) < 0) {

JOptionPane.showMessageDialog(null, "Expenses cannot be a negative number. \nPlease try again.",

"Input Error", JOptionPane.ERROR\_MESSAGE);

isExpenseValid = false;

// if the expense already exists but the user wants to change its contents

} else if (DisplaySpreadsheet.getRowIndex() >= 0 && DisplaySpreadsheet.getEditedClicked() == true) {

DisplaySpreadsheet.getSpreadsheet().setValueAt(Double.parseDouble(expenseJan.getText()), DisplaySpreadsheet.getRowIndex(), 1);

DisplaySpreadsheet.getSpreadsheet().setValueAt(Double.parseDouble(expenseFeb.getText()), DisplaySpreadsheet.getRowIndex(), 2);

DisplaySpreadsheet.getSpreadsheet().setValueAt(Double.parseDouble(expenseMar.getText()), DisplaySpreadsheet.getRowIndex(), 3);

DisplaySpreadsheet.getSpreadsheet().setValueAt(Double.parseDouble(expenseApr.getText()), DisplaySpreadsheet.getRowIndex(), 4);

DisplaySpreadsheet.getSpreadsheet().setValueAt(Double.parseDouble(expenseMay.getText()), DisplaySpreadsheet.getRowIndex(), 5);

DisplaySpreadsheet.getSpreadsheet().setValueAt(Double.parseDouble(expenseJun.getText()), DisplaySpreadsheet.getRowIndex(), 6);

DisplaySpreadsheet.getSpreadsheet().setValueAt(Double.parseDouble(expenseJul.getText()), DisplaySpreadsheet.getRowIndex(), 7);

DisplaySpreadsheet.getSpreadsheet().setValueAt(Double.parseDouble(expenseAug.getText()), DisplaySpreadsheet.getRowIndex(), 8);

DisplaySpreadsheet.getSpreadsheet().setValueAt(Double.parseDouble(expenseSept.getText()), DisplaySpreadsheet.getRowIndex(), 9);

DisplaySpreadsheet.getSpreadsheet().setValueAt(Double.parseDouble(expenseOct.getText()), DisplaySpreadsheet.getRowIndex(), 10);

DisplaySpreadsheet.getSpreadsheet().setValueAt(Double.parseDouble(expenseNov.getText()), DisplaySpreadsheet.getRowIndex(), 11);

DisplaySpreadsheet.getSpreadsheet().setValueAt(Double.parseDouble(expenseDec.getText()), DisplaySpreadsheet.getRowIndex(), 12);

} else {

// otherwise, the expense does not exist and needs to be added to the spreadsheet

expense.setJanuary(Double.parseDouble(expenseJan.getText()));

expense.setFebruary(Double.parseDouble(expenseFeb.getText()));

expense.setMarch(Double.parseDouble(expenseMar.getText()));

expense.setApril(Double.parseDouble(expenseApr.getText()));

expense.setMay(Double.parseDouble(expenseMay.getText()));

expense.setJune(Double.parseDouble(expenseJun.getText()));

expense.setJuly(Double.parseDouble(expenseJul.getText()));

expense.setAugust(Double.parseDouble(expenseAug.getText()));

expense.setSeptember(Double.parseDouble(expenseSept.getText()));

expense.setOctober(Double.parseDouble(expenseOct.getText()));

expense.setNovember(Double.parseDouble(expenseNov.getText()));

expense.setDecember(Double.parseDouble(expenseDec.getText()));

}

} catch (NumberFormatException e) {

JOptionPane.showMessageDialog(null,

"Please enter a number value for the expenses.",

"Input Error", JOptionPane.ERROR\_MESSAGE);

isExpenseValid = false;

}

// if the user clicked on Edit an Expense button, then the expense is updated, close the window afterwards

if (isExpenseValid && DisplaySpreadsheet.getEditedClicked() == true) {

DisplaySpreadsheet.setEditedClicked(false);

JOptionPane.showMessageDialog(null,

"Expense Successfully Updated!",

"Expense updated", JOptionPane.INFORMATION\_MESSAGE);

dispose();

// otherwise the user clicked on Add an Expense, and the expense is added, close the window afterwards

} else if (isExpenseValid && DisplaySpreadsheet.getEditedClicked() == false) {

DisplaySpreadsheet.addRowToSpreadsheet(expense);

JOptionPane.showMessageDialog(null,

"Expense Successfully Added!",

"Expense added", JOptionPane.INFORMATION\_MESSAGE);

dispose();

}

}

});

add(addExpense);

exitButton = new JButton("Back to Table");

exitButton.setBounds(265, 440, 125, 25);

exitButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent arg0) {

// close the window

dispose();

}

});

add(exitButton);

}

public static void main(String[] args) {

// TODO Auto-generated method stub

java.awt.EventQueue.invokeLater(new Runnable() {

public void run() {

new EditExpense().setVisible(true);

}

});

}

}